# **Name: Rajesh A**

**SUPERSET\_ID: 6384182**

# **MOQ HANDSON**

# **1.Write Testable Code with Moq**

# Unit Testing with Moq and NUnit in C#

## Objective:

To demonstrate how to write testable code using Dependency Injection (DI) and perform unit testing using Moq and NUnit frameworks in a .NET C# environment.

## Tools & Technologies Used:

- .NET SDK (v9.0)  
- Visual Studio Code  
- Moq (v4.20.69)  
- NUnit (v4.0.0)  
- NUnit Test Adapter

## Project Structure:

1. CustomerCommLib (Class Library)  
2. CustomerComm.Tests (Unit Test Project)

## Step-by-Step Implementation:

### 1. Create Class Library - CustomerCommLib

File: MailSender.cs

using System.Net;  
using System.Net.Mail;  
  
namespace CustomerCommLib  
{  
 public interface IMailSender  
 {  
 bool SendMail(string toAddress, string message);  
 }

public class MailSender : IMailSender  
 {  
 public bool SendMail(string toAddress, string message)  
 {  
 MailMessage mail = new MailMessage();  
 SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");  
  
 mail.From = new MailAddress("your\_email@gmail.com");  
 mail.To.Add(toAddress);  
 mail.Subject = "Test Mail";  
 mail.Body = message;  
  
 SmtpServer.Port = 587;  
 SmtpServer.Credentials = new NetworkCredential("username", "password");  
 SmtpServer.EnableSsl = true;  
 SmtpServer.Send(mail);  
  
 return true;  
 }  
 }  
  
 public class CustomerComm  
 {  
 IMailSender \_mailSender;  
  
 public CustomerComm(IMailSender mailSender)  
 {  
 \_mailSender = mailSender;  
 }  
  
 public bool SendMailToCustomer()  
 {  
 return \_mailSender.SendMail("cust123@abc.com", "Some Message");  
 }  
 }  
}

### 2. Create Unit Test Project - CustomerComm.Tests

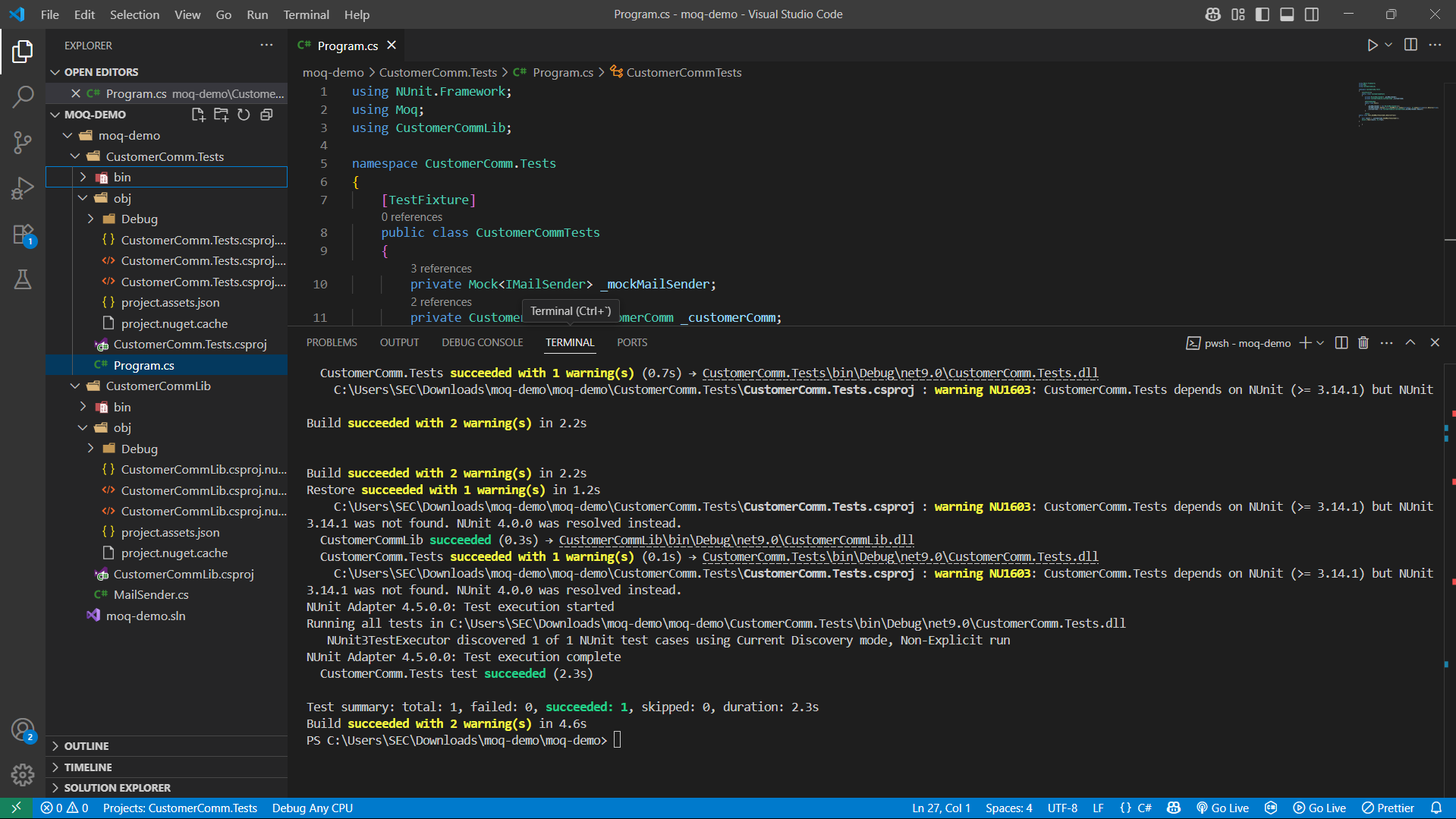
File: Program.cs

using NUnit.Framework;  
using Moq;  
using CustomerCommLib;  
  
namespace CustomerComm.Tests  
{  
 [TestFixture]  
 public class CustomerCommTests  
 {  
 private Mock<IMailSender> \_mockMailSender;  
 private CustomerCommLib.CustomerComm \_customerComm;  
  
 [OneTimeSetUp]  
 public void Init()  
 {  
 \_mockMailSender = new Mock<IMailSender>();  
 \_mockMailSender.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>())).Returns(true);  
 \_customerComm = new CustomerCommLib.CustomerComm(\_mockMailSender.Object);  
 }  
  
 [Test]  
 public void Test\_SendMailToCustomer\_ReturnsTrue()  
 {  
 bool result = \_customerComm.SendMailToCustomer();  
 Assert.That(result, Is.True);  
 }  
 }  
}

### 3. Terminal Commands to Execute the Project:

dotnet restore  
dotnet build  
dotnet test

### 4. Output :



1. **Mock file object for Unit Tests**

# Unit Testing with Mocked File System in C# using Moq :

## Objective:

To perform unit testing for file system operations by mocking Directory.GetFiles using Moq.

## Tools & Technologies Used:

- .NET SDK (v9.0)  
- Visual Studio Code  
- Moq  
- NUnit  
- NUnit Test Adapter

## Step-by-Step Implementation:

### 1. Create Class Library - MagicFilesLib

Commands:

dotnet new classlib -n MagicFilesLib  
cd MagicFilesLib

Create a file named `DirectoryExplorer.cs` and paste the following:

using System.Collections.Generic;  
using System.IO;  
  
namespace MagicFilesLib  
{  
 public interface IDirectoryExplorer  
 {  
 ICollection<string> GetFiles(string path);  
 }  
  
 public class DirectoryExplorer : IDirectoryExplorer  
 {  
 public ICollection<string> GetFiles(string path)  
 {  
 string[] files = Directory.GetFiles(path);  
 return files;  
 }  
 }  
}

### 2. Create Unit Test Project - DirectoryExplorer.Tests

dotnet new classlib -n DirectoryExplorer.Tests  
cd DirectoryExplorer.Tests

Add required packages:

dotnet add package NUnit  
dotnet add package Moq  
dotnet add package NUnit3TestAdapter  
dotnet add package Microsoft.NET.Test.Sdk

Add reference to main library:

dotnet add reference ../MagicFilesLib/MagicFilesLib.csproj

Replace `Class1.cs` with a file `Program.cs` and paste the following:

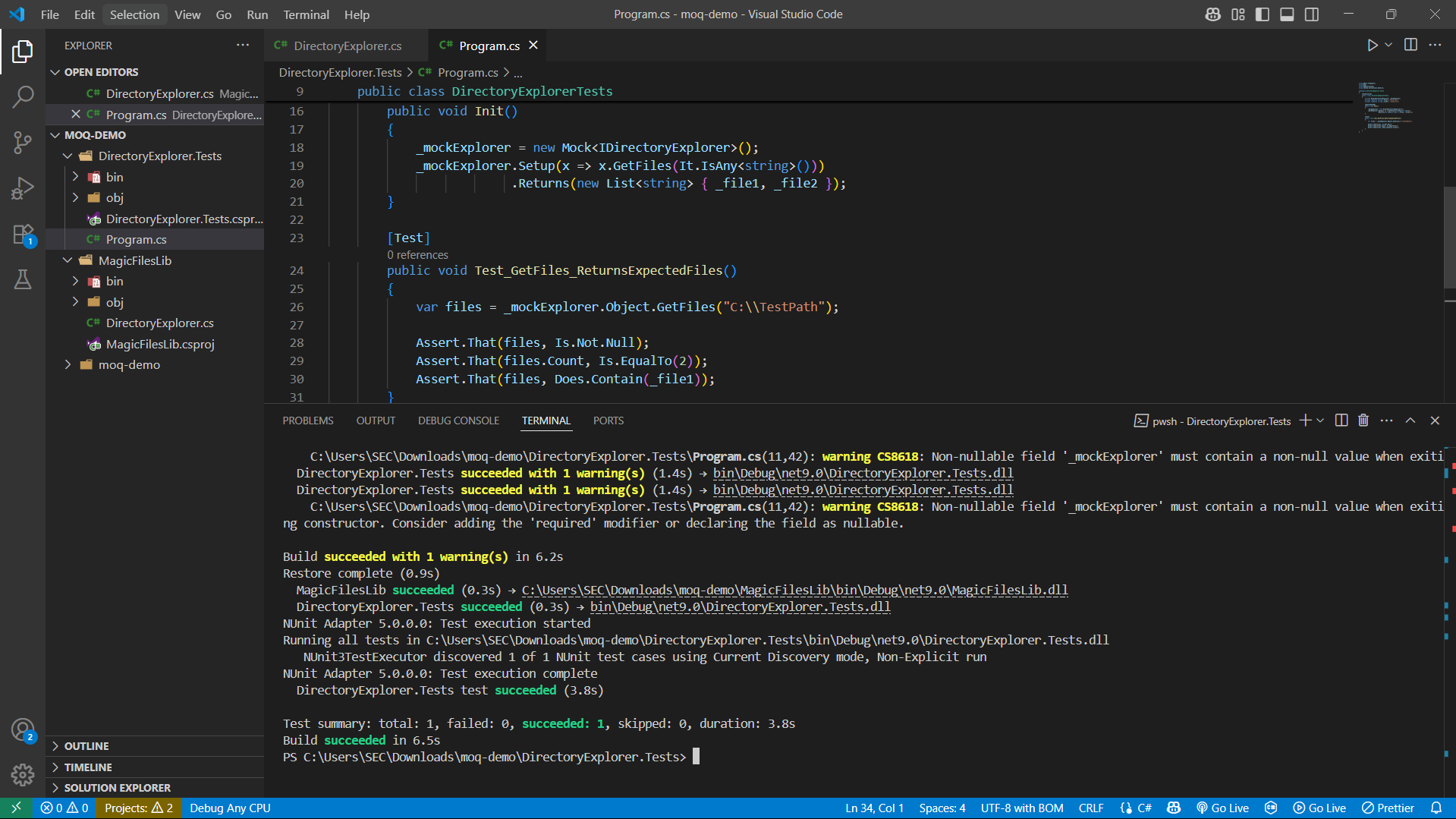
using NUnit.Framework;  
using Moq;  
using MagicFilesLib;  
using System.Collections.Generic;  
  
namespace DirectoryExplorer.Tests  
{  
 [TestFixture]  
 public class DirectoryExplorerTests  
 {  
 private Mock<IDirectoryExplorer> \_mockExplorer;  
 private readonly string \_file1 = "file.txt";  
 private readonly string \_file2 = "file2.txt";  
  
 [OneTimeSetUp]  
 public void Init()  
 {  
 \_mockExplorer = new Mock<IDirectoryExplorer>();  
 \_mockExplorer.Setup(x => x.GetFiles(It.IsAny<string>()))  
 .Returns(new List<string> { \_file1, \_file2 });  
 }  
  
 [Test]  
 public void Test\_GetFiles\_ReturnsExpectedFiles()  
 {  
 var files = \_mockExplorer.Object.GetFiles("C:\TestPath");  
  
 Assert.That(files, Is.Not.Null);  
 Assert.That(files.Count, Is.EqualTo(2));  
 Assert.That(files, Does.Contain(\_file1));  
 }  
 }  
}

### 3. Run the Test

From the root of the project directory:

dotnet restore  
dotnet build  
dotnet test

### 4. Paste Your Terminal Output Below:



**3 Mock database for Unit Tests**

# **Mock File Object for Unit Tests**

## Objective:

To perform unit testing for file system operations by mocking Directory.GetFiles() using **Moq** and **NUnit**, without depending on actual disk operations.

## Tools & Technologies Used:

- .NET SDK (v9.0)  
- Visual Studio Code  
- Moq  
- NUnit  
- NUnit Test Adapter

## Step-by-Step Implementation:

### 1. Create Class Library - MagicFilesLib

Commands:

dotnet new classlib -n MagicFilesLib  
cd MagicFilesLib

Create a file named `DirectoryExplorer.cs` and paste the following:

using System.Collections.Generic;

using System.IO;

namespace MagicFilesLib

{

public interface IDirectoryExplorer

{

ICollection<string> GetFiles(string path);

}

public class DirectoryExplorer : IDirectoryExplorer

{

public ICollection<string> GetFiles(string path)

{

string[] files = Directory.GetFiles(path);

return files;

}

}

}

### 2. Create Unit Test Project - DirectoryExplorer.Tests

dotnet new classlib -n DirectoryExplorer.Tests  
cd DirectoryExplorer.Tests

using NUnit.Framework;

using Moq;

using MagicFilesLib;

using System.Collections.Generic;

namespace DirectoryExplorer.Tests

{

[TestFixture]

public class DirectoryExplorerTests

{

private Mock<IDirectoryExplorer> \_mockExplorer;

private readonly string \_file1 = "file.txt";

private readonly string \_file2 = "file2.txt";

[OneTimeSetUp]

public void Init()

{

\_mockExplorer = new Mock<IDirectoryExplorer>();

\_mockExplorer.Setup(x => x.GetFiles(It.IsAny<string>()))

.Returns(new List<string> { \_file1, \_file2 });

}

[Test]

public void Test\_GetFiles\_ReturnsExpectedFiles()

{

var files = \_mockExplorer.Object.GetFiles(@"C:\TestPath");

Assert.That(files, Is.Not.Null);

Assert.That(files.Count, Is.EqualTo(2));

Assert.That(files, Does.Contain(\_file1));

}

}

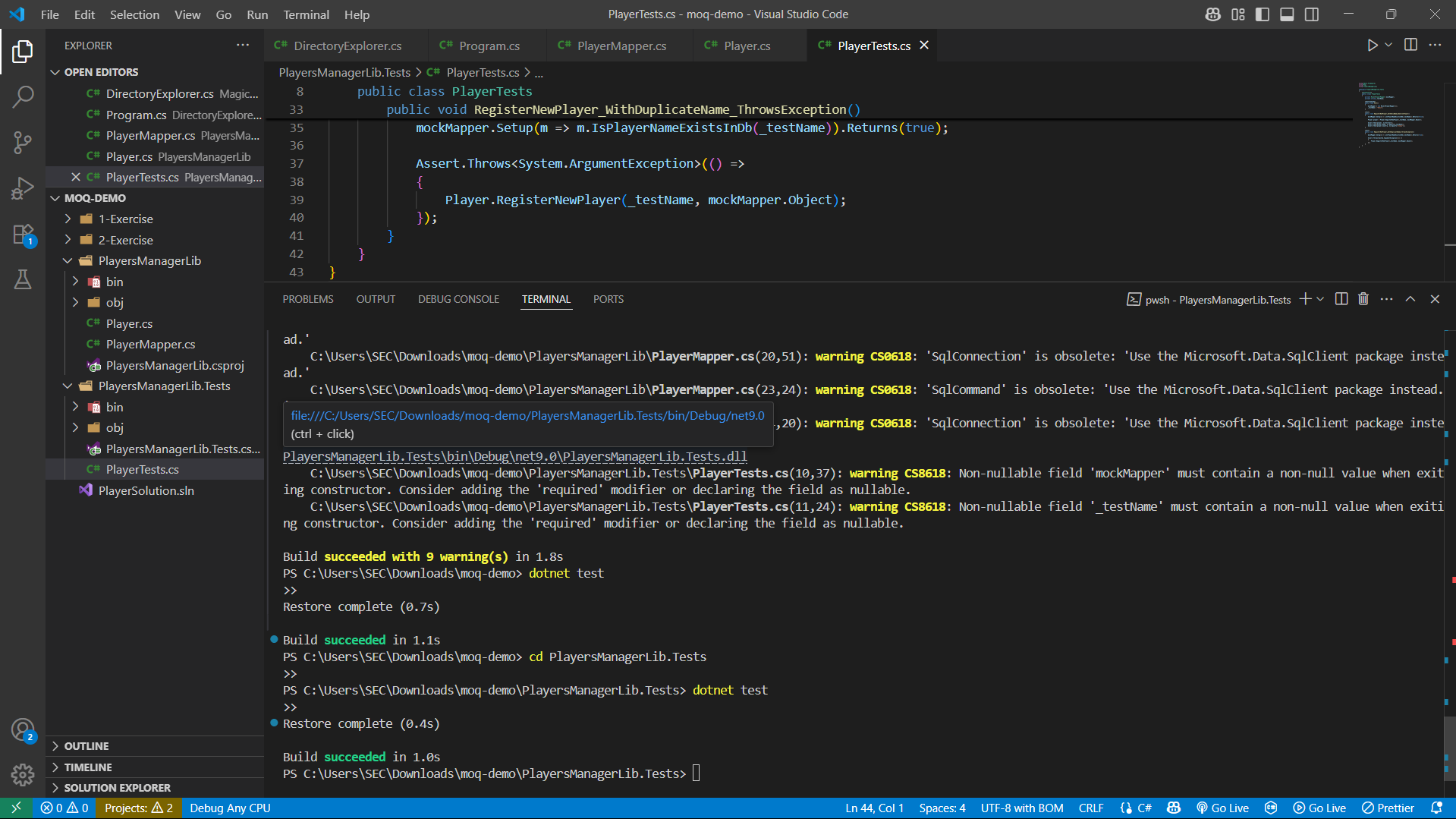
}

### 3. Run the Test

From the root of the project directory:

dotnet restore  
dotnet build  
dotnet test

### 3.Paste Your Terminal Output Below:



**Thank you**